Introduction to Computational Physics

PHYS 250 (Autumn 2018) – Lecture 2

David Miller

Department of Physics and the Enrico Fermi Institute
University of Chicago

October 4, 2018

Outline

Computational Physics (PHYS 250)

Course Description PHYS 250 (link to Course Catalog)

This course introduces the use of computers in the physical sciences. After an introduction to programming basics, we cover numerical solutions to fundamental types of problems, including cellular automatons, artificial neural networks, computer simulations of complex systems, and finite element analysis. Additional topics may include an introduction to graphical programming, with applications to data acquisition and device control.

There are an infinite number of paths that we might follow and still not deviate from this description. I therefore would like to lay out some of the principles that will guide me, and us, in how we navigate through those many possibilities.

Outline

1. Introduction

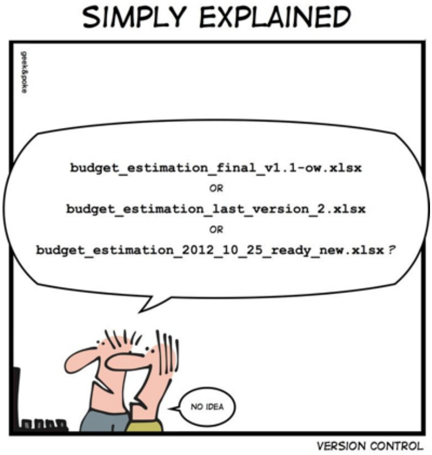
2. Software

Version control

- The most important message of this slide is simple...**Use a software version control system for all of your code**
  - And that means now...not tomorrow or next week
  - Because if you wait until you need it, it will be too late

Version control

- The most important message of this slide is simple...**Use a software version control system for all of your code**
    - And that means now...not tomorrow or next week
    - Because if you wait until you need it, it will be too late



A brief history of version control

- **The first version control systems were designed to be used on large systems where everyone logged into the same machine**
  - They tracked code on the same filesystem where it lived (e.g., in a subdirectory)
  - SCCS and RCS are examples
- **Then client-server systems were developed, so that developers could work on their own machines**
  - Checking code into a central server to share and collaborate
  - CVS and SVN are examples
- **More recently distributed version control systems have arisen**
  - These are decentralised, so everyone has a complete copy of the repository
  - Gives a lot of freedom to developers to share and merge as they like, so liked very much by the open source community
  - git, mercurial and bit keeper are examples

`git`, `GitHub`, & `GitLab`

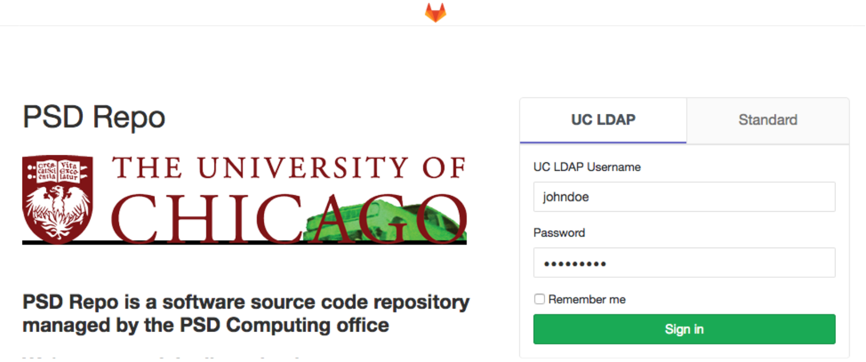
https://git-scm.com, *https://github.com*, *https://about.gitlab.com*

- **`git` is the most popular open source version control system**
    - can host huge projects (Linux Kernel, LHC experiment software, etc)
    - scales very well and it's extremely fast and powerful
    - very flexible (= complex)
- **Distributed version control systems (`git`) are great, but they're made even better by using a social coding site (`GitHub` or `GitLab`)**
- **These sites allow developers:**
    - browse code easily
    - compare different versions
    - take copies (a.k.a. fork)
    - offer patches back to upstream repositories
        - And discuss and review these patches before acceptance
    - even build websites
- **The best known social coding site is GitHub, but there are others, e.g. BitBucket and GitLab**
    - Familiarity with **`git`/`GitHub`/`GitLab`** will serve you well, trust me

GitHub & GitLab resources

GitHub is a free resource as long as your code remains public (you can pay for private repositories). The Physical Sciences Division (PSD) at UChicago hosts a **private** **GitLab** repository.

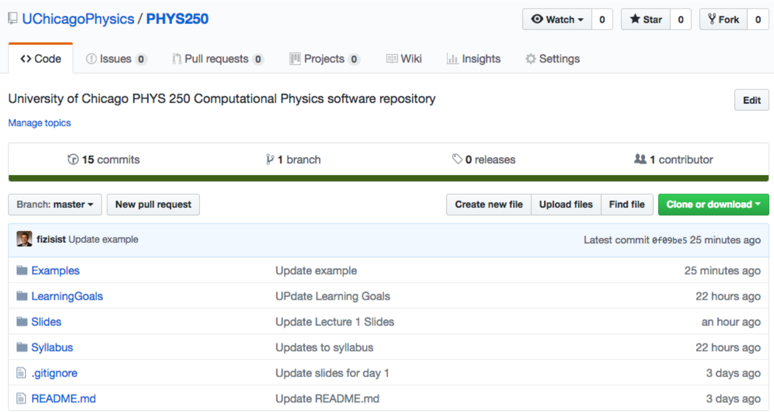
- https://psdcomputing.uchicago.edu/page/psd-repo



PHYS 250 **GitHub**

https://github.com/UChicagoPhysics/PHYS250

Course materials are hosted in the **GitHub** `UChicagoPhysics` repository



- Slides (e.g. *these!*), syllabi, learning goals, and code examples
- Stable versions will be cross-posted to Canvas as well.
- Homework submission will be done via **GitHub** (*instructions to come*)

Linux "shell"

- We will be using an interface to Linux called a "shell"
- It is a command-line interpreter : you type, it executes
- Two major options are `bash` (as in, smash) and `csh` (like "sea shell", modern version is "tcsh", "tea sea shell")
  - Only real difference: environment variables syntax
  - `bash: export X=value`
  - `csh: setenv X value`

Shell basics

Listing directory contents : `ls`, like "list"

```
> ls
Examples/ LearningGoals/ README.md Slides/
Syllabus/ global.sty
```

Copy: `cp`

```
> cp stuff.txt stuff1.txt
```

Where am I?: `pwd`, `cd`

```
> pwd
/ComputationalPhysics/PHYS250/PHYS250-Autumn2018
> ls
Examples/ LearningGoals/ README.md Slides/ Syllabus/
> cd Examples/
> ls
HelloGaussian.ipynb
HelloGaussian.py
Introduction_to_Jupyter_Notebooks_and_Python.ipynb
```

Hello world!

Interactive in the python interpreter

```
python
>>> print "hello world"
hello
>>> CTRL-d  # to exit python
```

From a script (containing the above print line):

```
python helloworld.py
```

Self-running script:

```
#!/usr/bin/env python
# This script prints hello to the screen
print "hello world"
```

```
chmod +x helloworld.py
./helloworld.py
hello world
```

Lists (I)

In my opinion, python's great advantage is **list comprehension**.

List basics

```
v = []     # empty list
v = list()   # empty list
v = [ 1, 2, 4, 5 ]  ; v = [ 'a', 'b', 'c' ]
v = range(4,10,2) # results in [ 4,6,8 ]
v = [ 4, 2.5, 'Hi', [ 1,3,5 ] ]  # can mix types
```

Append elements

```
>>> v.append( 70 )
>>> print v
```

Concatenation

```
>>> v += [ 'some', 'more', 'elements' ]
>>> v  # shows the object
```

Removal of elements

```
>>> v.remove(2.5)
>>> del v[0]
```

Lists (II)

Element acces read/write

```
>>> v[0]
'hi'
>>> v[0] = 'hey'
>>> v[-1] # last element. Negative = count from the en
>>> v[1:3] # subrange by index (start index, one-beyon
```

Test if an element is in a list (or not)

```
>>> if 4 in v:
...   print "Found it"
Found it
>>> if 200 not in v:
...   print "Not found"
Not found
```

`for` and `while` loops

The `for` statement iterates through a collection, iterable object or generator function.

The `while` statement merely loops until a condition is `False`.

Iterate over list

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Iterate using built-in `range` function

```
for x in range(0, 3):
  print "We're on time %d" % (x)
```

Iterate until a condition is met

```
count = 0
while count < 5:
  print(count)
  count += 1  # Same as: count = count + 1
```

Putting lists and loops together is amazing (and complex)

Filter one list into another (the "old" way)

```
newlist = []
for i in oldlist:
    if filter(i):
        newlist.append(function(i))
```

List comprehension (the "pythonic" way)

```
newlist = [function(i) for i in oldlist if filter(i)]
```

where `filter` and `function` just perform "some" operation on the list elements. Basically, the syntax is:

```
[ expression for item in list if conditional ]
```

and this replaces:

```
for item in list:
  if conditional:
    expression
```

Useful list comprehension

Filter one list into another (the "old" way)

```
>>> v = [ x**2 for x in range(10) if x % 3 == 0 ]
>>> v
[0, 9, 36, 81]
```

List comprehension (the "pythonic" way)

```
newlist = [function(i) for i in oldlist if filter(i)]
```

where `filter` and `function` just perform "some" operation on the list elements. Basically, the syntax is:

```
[ expression for item in list if conditional ]
```

and this replaces:

```
for item in list:
  if conditional:
    expression
```

Hello Gaussian!

Basic but useful code example

```
import numpy as np
import matplotlib.pyplot as plt

def p(x):
    return np.exp(-x**2)

#let's plot it
x = np.linspace(-3,3,100)
y = p(x)
plt.plot(x,y)
plt.show()
```

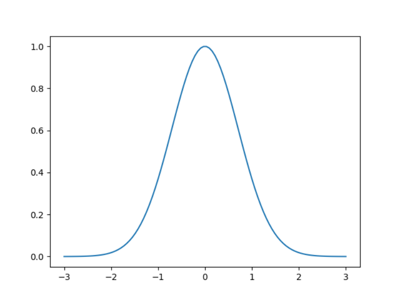
Hello Gaussian!

Basic but useful code example

```
import numpy as np
import matplotlib.pyplot as plt

def p(x):
    return np.exp(-x**2)

#let's plot it
x = np.linspace(-3,3,100)
y = p(x)
plt.plot(x,y)
plt.show()
```



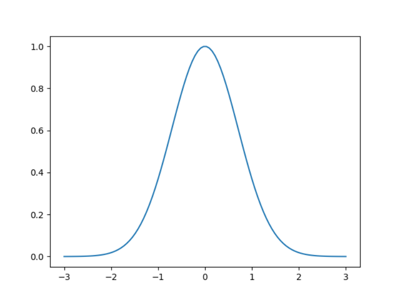
Hello Gaussian!

Basic but useful code example

```
import numpy as np
import matplotlib.pyplot as plt

def p(x):
    return np.exp(-x**2)

#let's plot it
x = np.linspace(-3,3,100)
y = p(x)
plt.plot(x,y)
plt.show()
```



But what about that `linspace` thingy? Google it! (numpy docs)

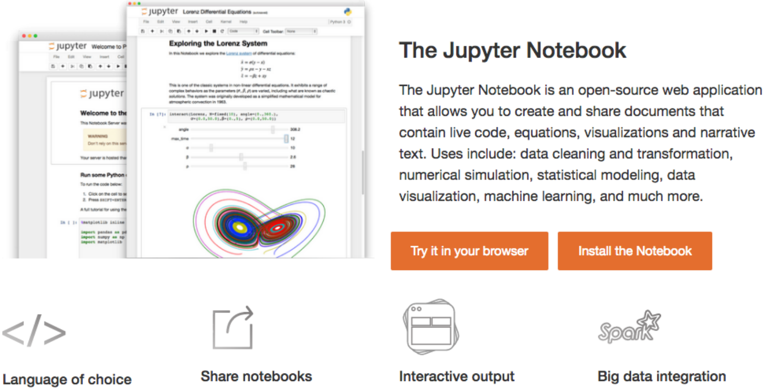
`numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)`

"Returns num evenly spaced samples, calculated over the interval [start, stop]."

Jupyter notebooks

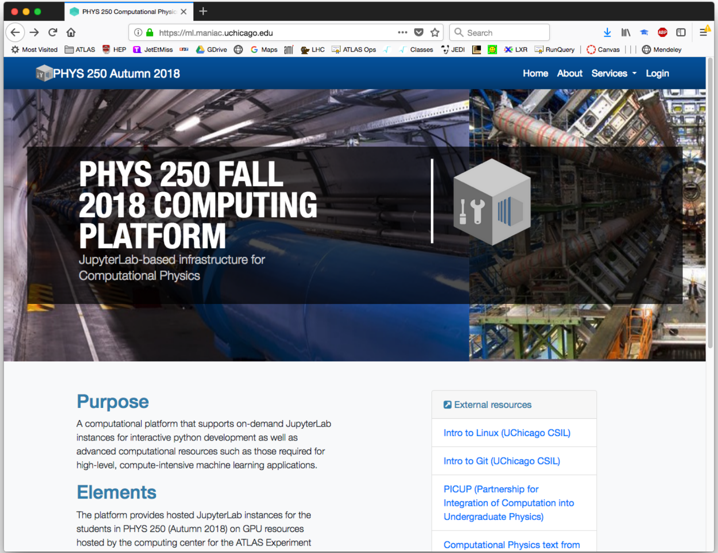
Interactive, web-based, integrated code and documentation environment

We will be following-up with more technical practice with python, but I want to introduce you to the resources that we'll be using this quarter for many of our examples and projects: Jupyter notebooks.



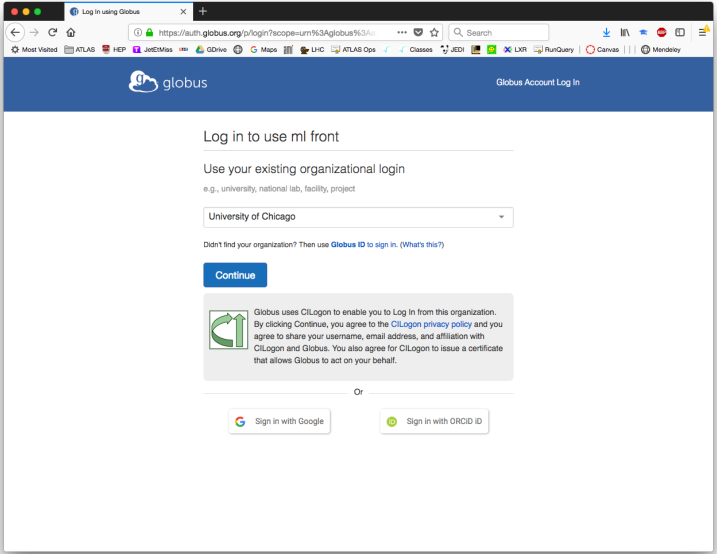
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



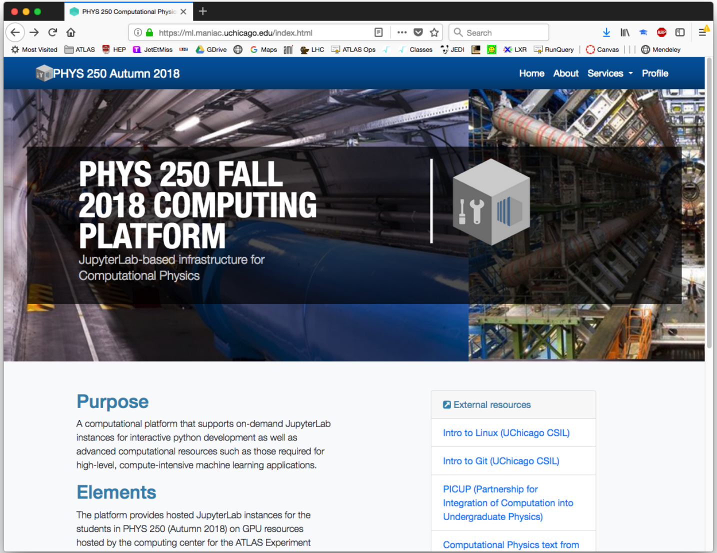
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



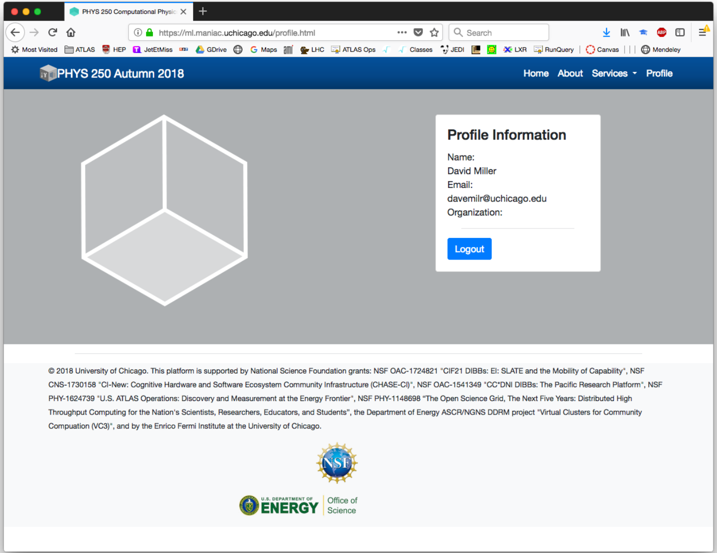
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



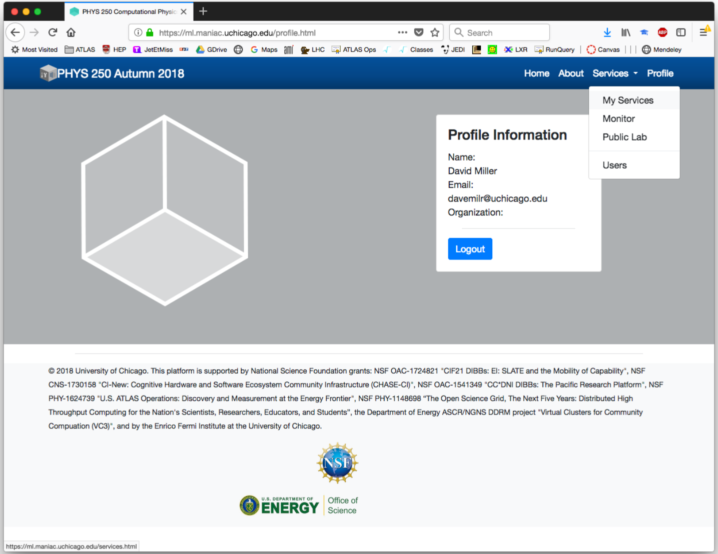
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



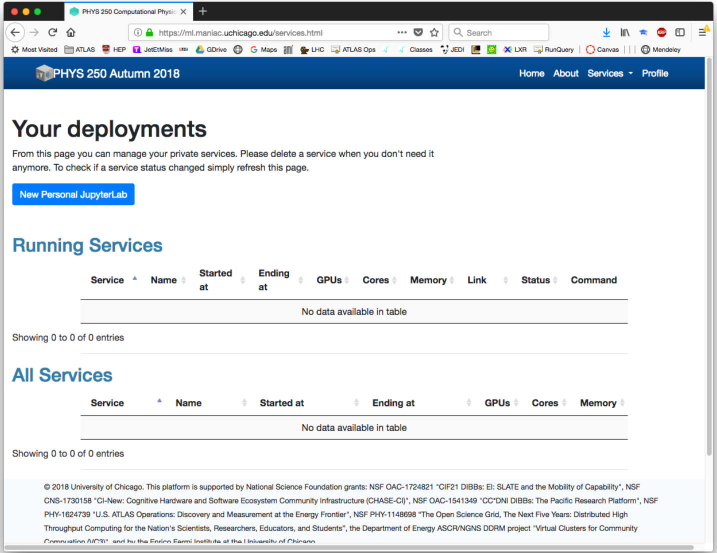
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



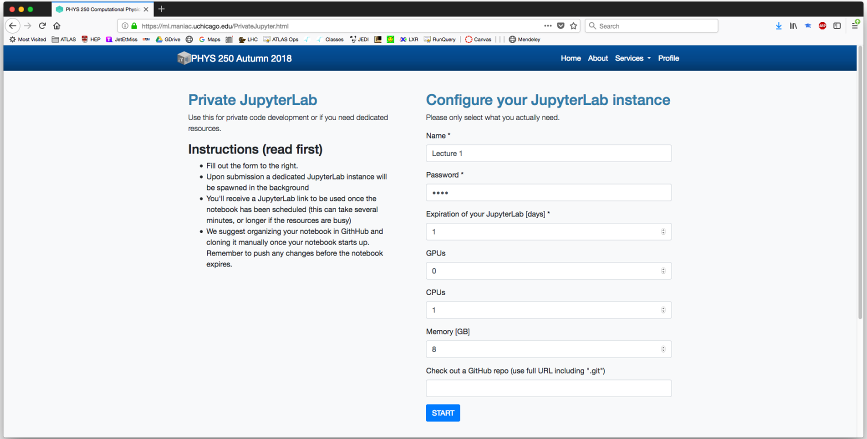
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



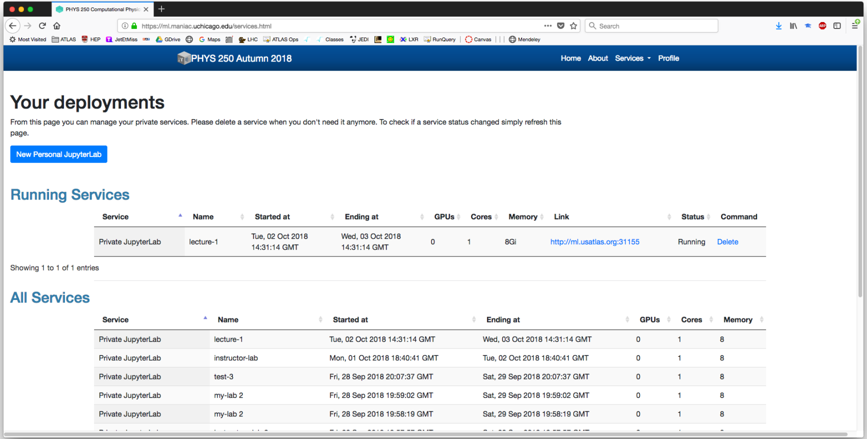
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.


PHYS 250 Autumn 2018

Home About Services Profile

Your deployments

From this page you can manage your private services. Please delete a service when you don't need it anymore. To check if a service status changed simply refresh this page.

New Personal JupyterLab

Running Services

Service	Name	Started at	Ending at	GPUs	Cores	Memory	Link	Status	Command
Private JupyterLab	lecture-1	Tue, 02 Oct 2018 14:31:14 GMT	Wed, 03 Oct 2018 14:31:14 GMT	0	1	8Gi	http://ml.usatlas.org:31155	Running	Delete

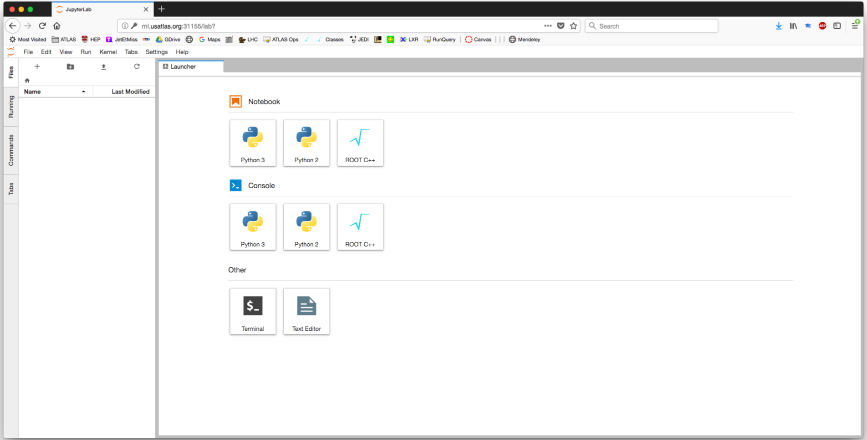
Showing 1 to 1 of 1 entries

All Services

Service	Name	Started at	Ending at	GPUs	Cores	Memory
Private JupyterLab	lecture-1	Tue, 02 Oct 2018 14:31:14 GMT	Wed, 03 Oct 2018 14:31:14 GMT	0	1	8
Private JupyterLab	instructor-lab	Mon, 01 Oct 2018 18:40:41 GMT	Tue, 02 Oct 2018 18:40:41 GMT	0	1	8
Private JupyterLab	test-3	Fri, 28 Sep 2018 20:07:37 GMT	Sat, 29 Sep 2018 20:07:37 GMT	0	1	8
Private JupyterLab	my-lab 2	Fri, 28 Sep 2018 19:59:02 GMT	Sat, 29 Sep 2018 19:59:02 GMT	0	1	8
Private JupyterLab	my-lab 2	Fri, 28 Sep 2018 19:58:19 GMT	Sat, 29 Sep 2018 19:58:19 GMT	0	1	8

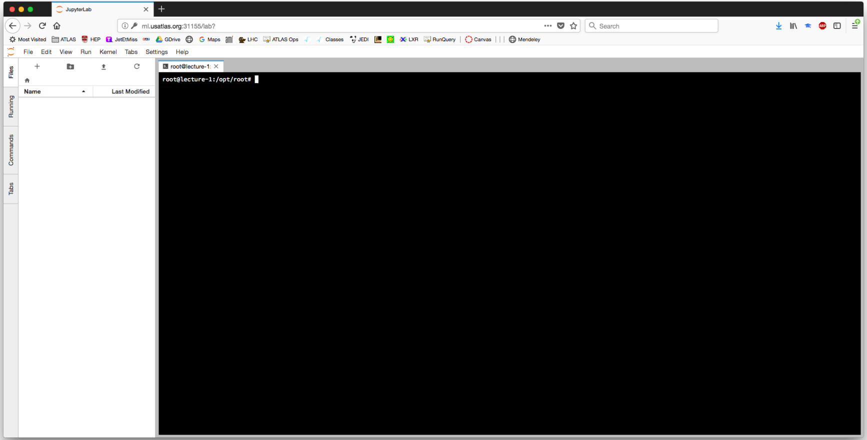
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



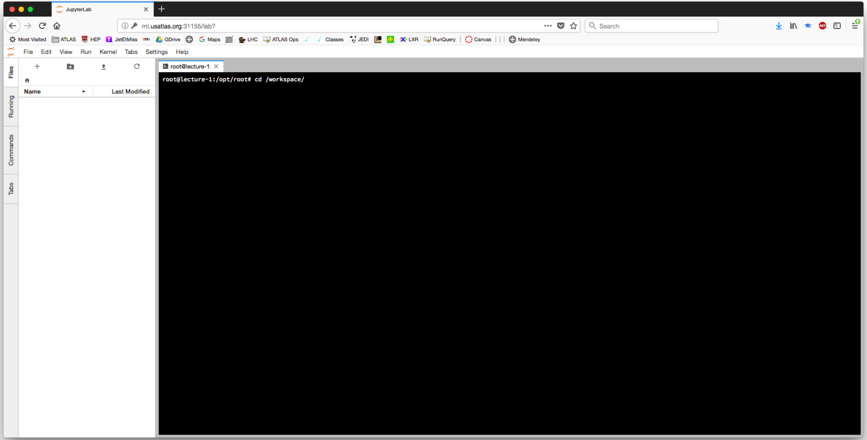
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



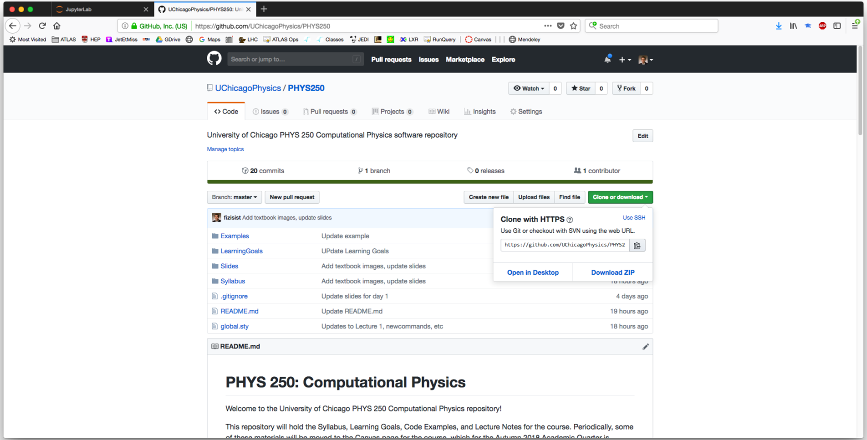
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



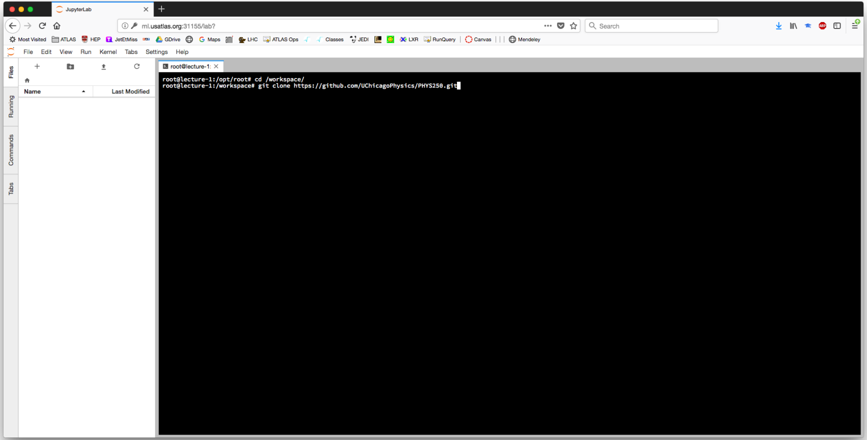
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.